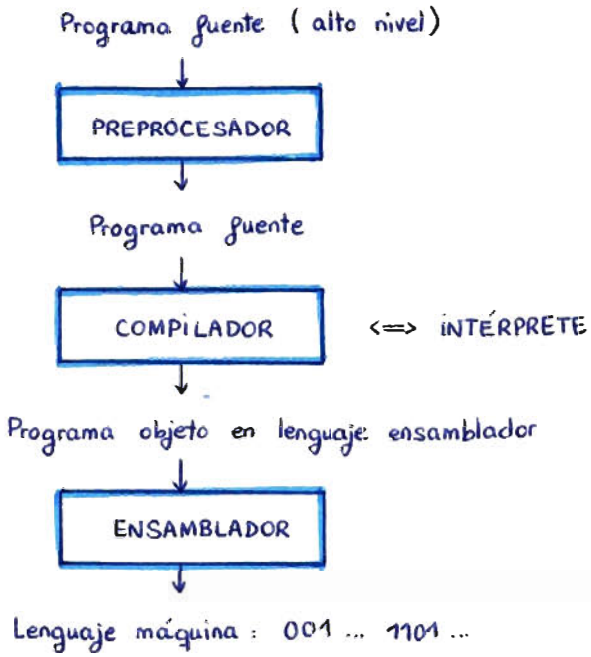
TEMA 1: INTRODUCCIÓN.

Un compilador es un programa que lee el código de un programa escrito en un determinado lenguaje (LENGUAJE FUENTE), por ejemplo, C o Pascal, y lo traduce a un programa equivalente en otro lenguaje (LENGUAJE OBJETO).

Los compiladores se pueden clasificar de distintas formas, por ejemplo, de una pasada (lee una sola vez el programa fuente) o de múltiples pasadas.



En una compilación típica, el compilador crea código en lenguaje ensamblador y éste es traducido a código máquina por un ensamblador.

* INTÉRPRETE Vs. COMPILADOR.

1) La ocupación de memoria es menor con el intérprete.

2) La depuración es más sencilla con un intérprete.

3) En el intérprete las instrucciones se van ejecutando a medida que se comprueba que son correctas. El intérprete, en lugar de producir un programa objeto como resultado de una traducción, realiza las operaciones que implica el programa fuente.

4) El intérprete está todo el tiempo en memoria; el compilador sólo durante la compilación.

5) El compilador pasa una vez por cada fase; el intérprete atraviesa cada fase por cada una de las instrucciones.

6) Intérprete clásico => Interpreta cada instrucción.
Intérprete actual => Traduce el leng. fuente a leng. intermedio e interpreta cada instrucción en leng. intermedio (L.I.)

7) El intérprete normalmente carece de fase de optimización.

8) La ejecución es más lenta con el intérprete.

9) Un intérprete normalmente es más pequeño que un compilador.

1. FASES DE UN COMPILADOR.

La compilación se divide en 2 partes:

a) **ANÁLISIS**: Divide el programa fuente en sus elementos componentes y crea una representación intermedia del programa fuente.
El análisis consta de 3 fases:

- ANÁLISIS LÉXICO: Comprueba si hay palabras mal escritas.
  Ejemplo: Regla para números enteros => E → dEld

- ANÁLISIS SINTÁCTICO: Comprueba que la sintaxis del programa sea correcta, es decir, que las estructuras estén bien construidas.
  Ejemplo: Regla while => S → while C do S end ;

- ANÁLISIS SEMÁNTICO: Principalmente comprueba los tipos y evita la duplicidad.
  Ejemplo: Asignación => id := E ;

b) SÍNTESIS: Construye el programa objeto deseado a partir de la representación intermedia generada en la etapa de análisis.

- GENERADOR DE CÓDIGO INTERMEDIO: Traduce el programa fuente correcto a un lenguaje intermedio de bajo nivel.

- OPTIMIZACIÓN DEL CÓDIGO (opcional).

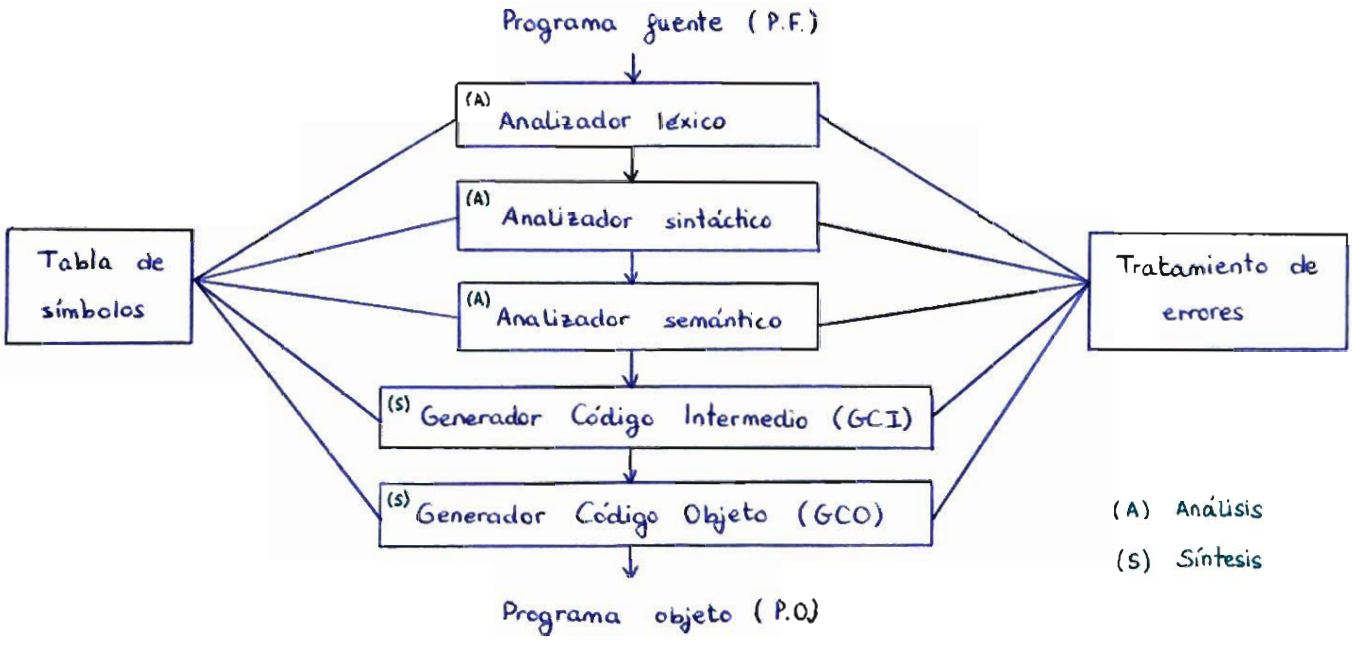
- GENERADOR DE CÓDIGO OBJETO: Recibe el código intermedio correcto y lo traduce a ensamblador.

Además, tenemos:

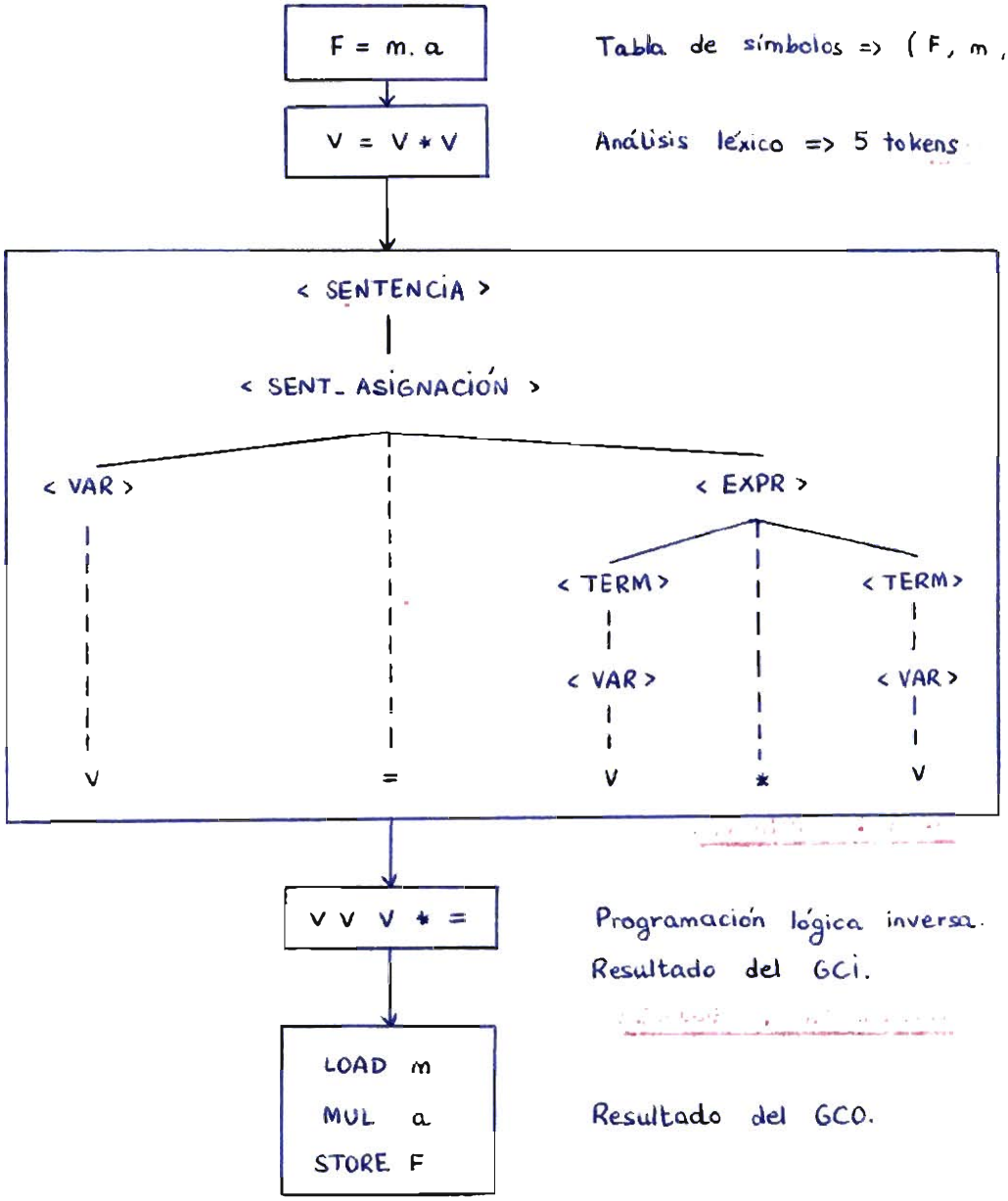
- TABLA DE SÍMBOLOS: Estructura de datos donde se almacena toda la información relativa a los identificadores del programa fuente que aparecen durante la compilación.

- TRATAMIENTO DE ERRORES: Existen 2 modalidades:
  - Encuentro un error y paro.
  - Encuentro un error y sigo (la idea es mostrar todos los errores de una vez).

En resumen:



Ejemplo:



Cada módulo recibe como entrada la salida del módulo anterior.

2. REPASO DE LENGUAJES Y GRAMÁTICAS.

- **ALFABETO**: Conjunto de símbolos válidos.

- **CADENA VÁLIDA**: Combinación de elementos del alfabeto.

- **LENGUAJE**: Conjunto de cadenas válidas.

- **GRAMÁTICA**: Conjunto de reglas que me permiten generar todas las cadenas válidas de un lenguaje. Es una representación finita de todas las cadenas de un lenguaje.

$$G = (N, T, P, S) \quad \text{(CHOMSKY)}$$

donde: N ≡ Conjunto de símbolos no terminales.
T ≡ " " " terminales.
P ≡ " " reglas de producción.
S ≡ Axioma de la gramática.

UTILIZAREMOS GRAMÁTICAS PARA EL ANÁLISIS LÉXICO Y SINTÁCTICO.

* TIPOS DE GRAMÁTICAS (SEGÚN CHOMSKY).

Chomsky clasificó las gramáticas en 4 grandes grupos (G_0, G_1, G_2, G_3), cada uno de los cuales incluye a los siguientes:

$$G_3 \subset G_2 \subset G_1 \subset G_0$$
$$L_3 \subset L_2 \subset L_1 \subset L_0$$

- **TIPO 0**: De estructura de grase.

$$\alpha \rightarrow \beta \quad , \quad \alpha \in (NUT)^{+} \; ; \; \beta \in (NUT)^{*}$$

$\alpha \in (NUT)^{+}$: No puede ser vacío.

$\beta \in (NUT)^{*}$: Símbolos No terminales y Terminales.

- **TIPO 1:** Sensibles al contexto.

$$\alpha A \beta \rightarrow \alpha \gamma \beta \quad , \quad A \in N \; ; \; \alpha, \beta \in (NUT)^* \; ; \; \gamma \in (NUT)^+$$

- **TIPO 2:** Independientes de contexto. (GCL = G. de Contexto Libre)

$$A \rightarrow \alpha \quad , \quad A \in N \; ; \; \alpha \in (NUT)^*$$

- **TIPO 3:** Gramáticas regulares. (GR)

$A \rightarrow aB$	$A \rightarrow Ba$
$A \rightarrow a$	$A \rightarrow a$
$S \rightarrow \lambda$	$S \rightarrow \lambda$
↓	↓
GLD	GLI
(Regulares por la dcha.)	(Regulares por la izq.)

, $A, B \in N \; ; \; a \in T \; ; \; S$ axioma.

El ANALIZADOR LÉXICO trabaja con gramáticas GLI. (regulares izq.)

El ANALIZADOR SINTÁCTICO comprueba que la sintaxis del lenguaje se puede generar con una GCL.

3. REPASO DE OTROS CONCEPTOS.

- **POTENCIA DE UN CONJUNTO:** Consiste en concatenar el conjunto i-veces.

$$A^0 = \{\lambda\}$$
$$A^1 = A$$
$$A^2 = A.A$$

- **CIERRE TRANSITIVO** (clausura positiva): $A^+ = \bigcup_{i=1}^{\infty} A^i$

- **CIERRE REFLEXIVO** (clausura o cierre): $A^* = \bigcup_{i=0}^{\infty} A^i$

} Se trata de unir el conjunto con sus potencias.

- **LENGUAJE GENERADO POR UNA GRAMÁTICA:**

$$L(G) = \{ x \;/\; (S \Rightarrow^* x) \text{ y } (x \in T^*) \}$$

x = sentencia.

≡ Cualquier cadena derivada a partir del axioma y cuyos símbolos son todos terminales.

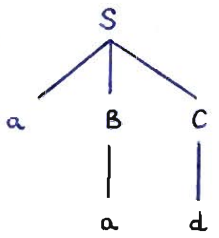
- **FORMA SENTENCIAL:**

$$D(G) = \{ \alpha \;/\; (S \Rightarrow^* \alpha) \text{ y } (\alpha \in (NUT)^*) \}$$

α = forma sentencial.

≡ Cualquier cadena derivada a partir del axioma y que está formada por símbolos terminales y no terminales.

Ejemplo:

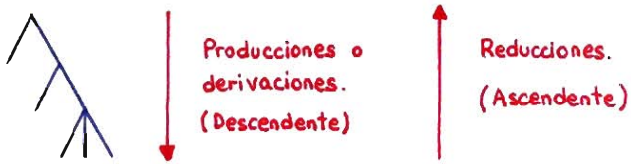


aBC = Forma sentencial.

aad = Sentencia.

- **DERIVACIONES:** Se dice que w es una derivación de v, o que v produce w, o que w se reduce a v, si existe una secuencia finita tal que:

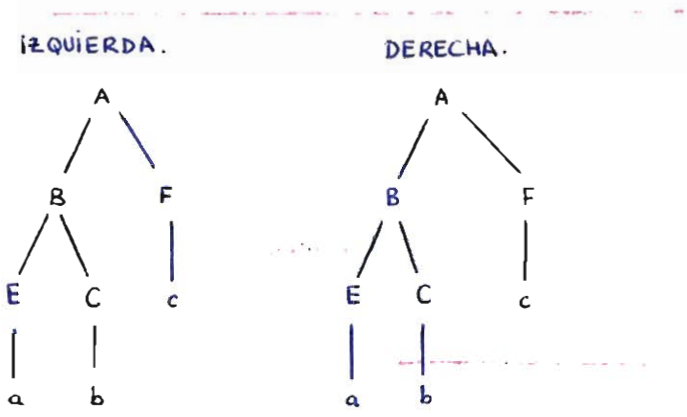
$$v = u_0 \rightarrow u_1 \rightarrow u_2 \ \ldots \ u_{n-1} \rightarrow u_n = w$$



- **Derivaciones por la izquierda** ⇒ Para expandir elijo siempre el símbolo no terminal más a la izquierda de la forma sentencial.
- **Derivaciones por la derecha** ⇒ Para expandir elijo siempre el símbolo no terminal más a la derecha de la forma sentencial.

Ejemplo:

① A → BF
② B → EC
③ E → a
④ C → b
⑤ F → c



Orden de las reglas aplicadas → 1 2 3 4 5 (izquierda) 1 5 2 4 3 (derecha)

ANALIZADOR SINTÁCTICO DESCENDENTE = PARSER

⇓

DERIVACIONES POR LA IZQUIERDA

- **RECURSIVIDAD:** Habrá que evitar la recursividad por la izquierda. Los otros tipos (por la derecha, autoimplicada) dan igual.

izq → $B \to Bfg$ ⇒ Para el análisis sintáctico descendente no puedo tener esta recursividad porque entraría en un bucle infinito. Hay que eliminarla.

$$A \to A\alpha_1 \mid A\alpha_2 \mid \dots \mid A\alpha_p \mid \beta_1 \mid \dots \mid \beta_q$$

↓

$$\left.\begin{array}{l} A \to \beta_1 \mid \dots \mid \beta_q \mid \beta_1 A' \mid \dots \mid \beta_q A' \\ A' \to \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_p \mid \alpha_1 A' \mid \dots \mid \alpha_p A' \end{array}\right\}$$

↓

Hemos eliminado la recursividad por la izquierda. Ahora tenemos recursividad por la derecha pero ésa no importa.

- **FACTORIZACIÓN POR LA IZQUIERDA:** El analizador sintáctico exige que cada regla empiece diferente.

$$A \rightarrow \alpha B_1 \mid \alpha B_2 \mid \dots \mid \alpha B_n \mid \gamma$$

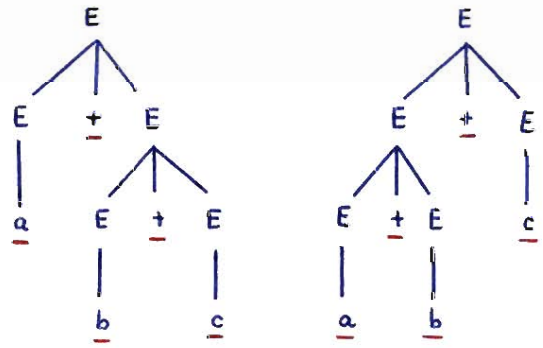
$$\downarrow$$

$$\left.\begin{array}{l} A \rightarrow \alpha R \mid \gamma \\ R \rightarrow B_1 \mid B_2 \mid \dots \mid B_n \end{array}\right\}$$

- **AMBIGÜEDAD:** Una gramática es ambigua si se puede obtener una cadena válida por dos árboles de derivación diferentes.
Un compilador no puede trabajar con ambigüedad. Hay que solucionarlo.

Ejemplo:

$E \rightarrow E + E$
$E \rightarrow a \mid b \mid c$

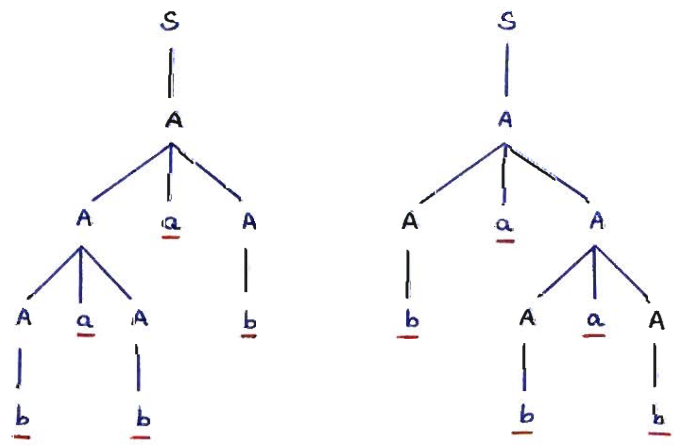


Existen 2 tipos de ambigüedad
- Por estructura del árbol.
- Por etiquetado.

Ejemplo: AMBIGÜEDAD ESTRUCTURADA.

$S \rightarrow A$
$A \rightarrow AaA$
$A \rightarrow b$

Cadena: babab

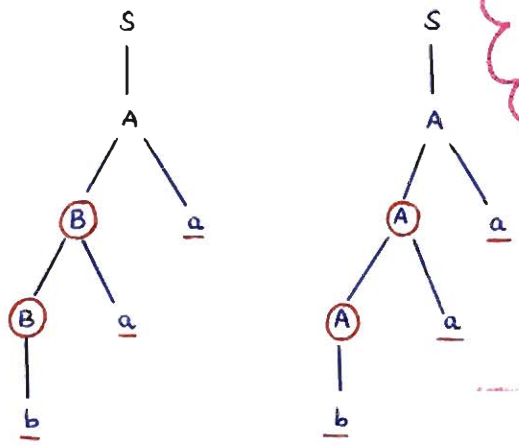


Ejemplo: AMBIGÜEDAD POR ETIQUETADO.

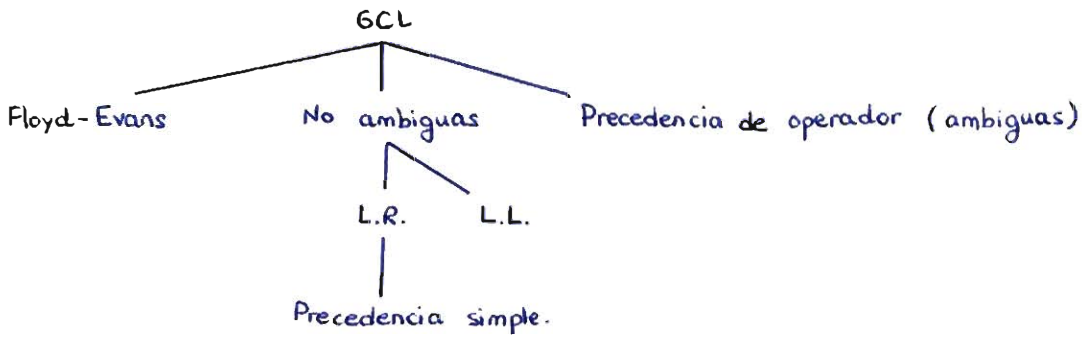
En una gramática regular sólo puede haber ambigüedad por etiquetado.

① S → A
② A → Ba
③ A → Aa
④ B → Ba
⑤ A → b
⑥ B → b

Cadena: baa



- JERARQUÍA DE GRAMÁTICAS:



- NOTACIÓN:

* BNF (Forma Normal de Backus): $A \to \alpha \mid \beta \mid \gamma$

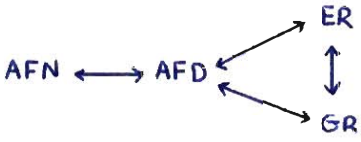
* EBNF (Forma Normal de Backus Extendida): $id \to \ell \{ \ell \mid d \}^*$

$$\Downarrow$$

$$\left.\begin{array}{l} id \to \ell R \\ R \to \ell R \mid dR \mid \lambda \end{array}\right\}$$

- PROPOSICIONES EQUIVALENTES:

1) L es un conjunto regular.
2) L se denota por una expresión regular (ER).
3) L puede ser definido por un AFN.
   ↳ Cualquier AFN puede transformarse en un AFD.



ANALIZADOR LÉXICO (AL) = AFD + ACCIONES SEMÁNTICAS.

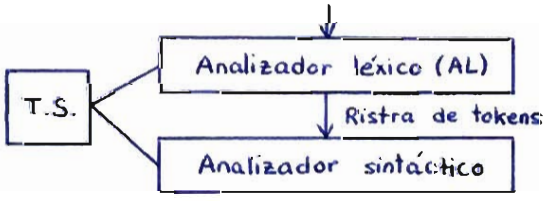
- AUTÓMATA FINITO:

$AF = (Q, Te, g, q_0, F)$

AFD $\Rightarrow$ $g: Q \times Te \to Q$
AFN $\Rightarrow$ $g: Q \times Te \to P(Q)$

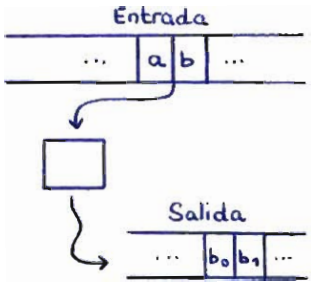
$$L(AF) = \{ t \mid t \in Te^*, (q_0, t) \vdash^* (q_i, \lambda), q_i \in F \}$$

Programa fuente



Mediante el AF sólo se sabe si el identificador es válido. Por tanto, el AL necesita algo más para devolver la ristra de tokens: => Acciones semánticas.
(A medida que vamos leyendo caracteres los vamos guardando para que el analizador sintáctico tenga algo con lo que trabajar).

- **TRADUCTOR FINITO (TF):** Reconoce y además traduce. Va escribiendo los tokens en una cinta de salida.



$$TF = (Q, T_e, T_s, \delta, q_0, F)$$

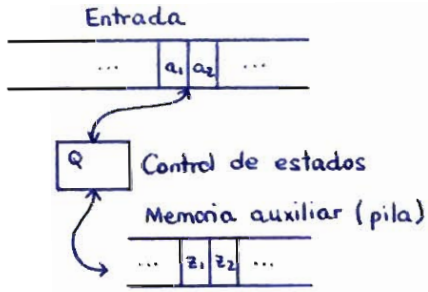
A. léxico

TFD $\Rightarrow \delta : Q \times (T_e \cup \lambda) \to Q \times T_s^*$

TFN $\Rightarrow \delta : Q \times (T_e \cup \lambda) \to P(Q \times T_s^*)$

$$T_s(TF) = \{ (t,s) \;/\; t \in T_e^*,\ s \in T_s^*,\ (q_0, t, \lambda) \vdash^* (q_F, \lambda, s) \}$$

- **AUTÓMATA CON PILA (AP):**



$$AP = (Q, T_e, T_p, \delta, q_0, z_0, F)$$

APD $\Rightarrow \delta : Q \times (T_e \cup \lambda) \times T_p \to Q \times T_p^*$

APN $\Rightarrow \delta : Q \times (T_e \cup \lambda) \times T_p \to P(Q \times T_p^*)$

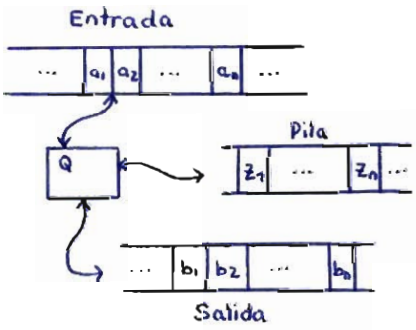
A. sintáctico

Usaremos esta opción. $\longrightarrow$

$$L(AP) = \{ t \;/\; t \in T_e^*,\ (q_0, t, z_0) \vdash^* (q_F, \lambda, \alpha) \}$$ (AP. POR ESTADO FINAL)

$$L(AP) = \{ t \;/\; t \in T_e^*,\ (q_0, t, z_0) \vdash^* (q', \lambda, \lambda) \}$$ (AP. POR VACIADO DE PILA)

- **TRADUCTOR DE PILA (TP):**



$$TP = (Q, T_e, T_s, T_p, \delta, q_0, z_0, F)$$

TPD $\Rightarrow \delta : Q \times (T_e \cup \lambda) \times T_p) \to Q \times T_p^* \times T_s^*$

TPN $\Rightarrow \delta : Q \times (T_e \cup \lambda) \times T_p) \to P(Q \times T_p^* \times T_s^*)$

$$T_s(TP) = \{ (t,s) \;/\; t \in T_e^*,\ s \in T_s^*,\ (q_0, t, z_0, \lambda) \vdash^* (q_F, \lambda, \alpha, s) \}$$ (TP. POR ESTADO FINAL)

$$T_s(TP) = \{ (t,s) \;/\; t \in T_e^*,\ s \in T_s^*,\ (q_0, t, z_0, \lambda) \vdash^* (q, \lambda, \lambda, s) \}$$ (TP. POR VACIADO DE PILA).

G. TIPO 2 $\Leftrightarrow$ AP, TP $\Leftrightarrow$ ANALIZADOR SINTÁCTICO.

G. TIPO 3 $\Leftrightarrow$ AFD, TFD $\Leftrightarrow$ " " LÉXICO.